Lab 4 – Binary Search Tree

Phần 1: Chuẩn bị

Để chuẩn bị thực hiện bài tập thực hành số 4 này, SV cần:

- Làm hết tất cả những bài tập trong Lab 3
- Tìm hiểu kiến thức về C++ Templates (xem thêm ở http://www.tutorialspoint.com/cplusplus/cpp_templates.htm)
- Nắm rõ các thuật toán trong tập slide **Chapter7 - Tree - 2009.pptx**

Phần 2: Hướng dẫn

Trong bài tập này, SV được hướng dẫn xây dựng một lớp tên BinarySearchTree.

Class này thể hiện được cấu trúc dữ liệu Cây nhị phân tìm kiếm (Binary Search Tree) mà SV đã được hướng dẫn.

1. main.cpp

Xem trong tập tin này, có đoạn mã nguồn sau:

```
	//data for bst B, need if you want to test your result in the Tutorial 4
	int B[] = {17, 10, 40, 31, 78, 19, 27, 33, 12, 6, 95, 71, 91};
	char* dataB[] = {"data17", "data10", "data40", "data31", "data78", "data19", "data27",
"data33", "data12", "data6", "data95", "data71", "data91"};

	// This code for insert key/data into the bstB
	BinarySearchTree<int, char*> *bstB = new BinarySearchTree<int, char*>();
	for (int i = 0; i < sizeof(B)/sizeof(int); i++)
		bstB->Insert(NodeEntry<int, char*>(B[i], dataB[i]));
	bstB->PrettyPrint();
```

Đoạn code này có nhiệm vụ khởi tạo một cây nhị phân tìm kiếm tên bstB

Thực hiện vòng lặp để chèn dữ liệu vào cây này

Dữ liệu được chèn là một NodeEntry có:

- Khoá (key) là một số nguyên (int), dùng để sắp xếp dữ liệu trong cây BST
- Dữ liệu (data) là một chuỗi kí tự (char*)

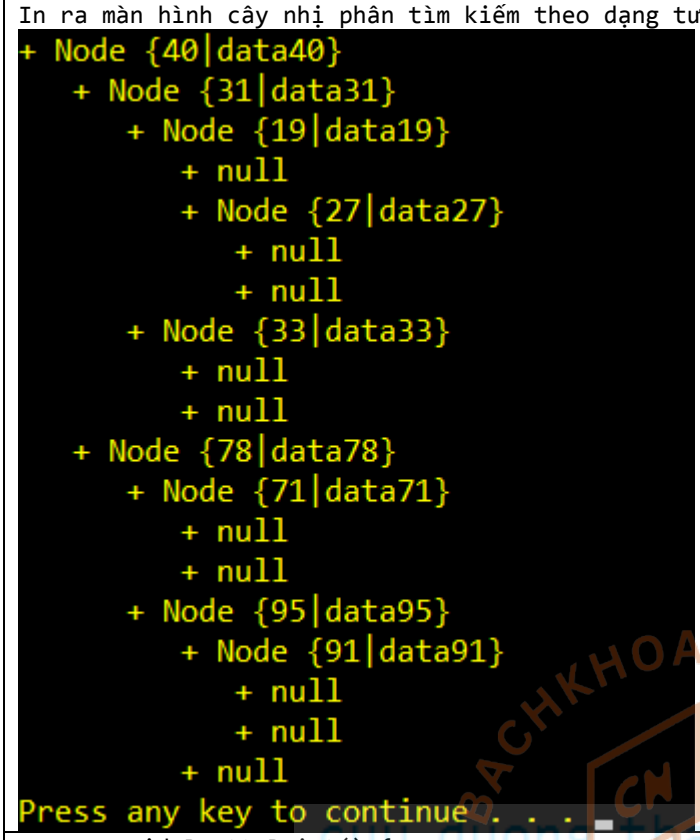
2. BinarySearchTree.h

Tập tin này chứa mã nguồn định nghĩa cấu trúc của lớp BinarySearchTree

SV cần thực hiện các phương thức của lớp này theo mô tả sau:

```
void PrettyPrint()
```

In ra màn hình cây nhị phân tìm kiếm theo dạng tương tự sau (theo input có sẵn trong main.cpp):




```
void PrettyPrint() {
	recursive_PrettyPrinted(_root, 0);
}

void recursive_PrettyPrinted(TreeNode<keyType, dataType> *const&subroot, int indent) {
	//Your code here
	for (int i = 0; i < indent; i++)
		if (i == indent-1) cout << "+ ";
		else cout << "    ";
	if (!subroot) {
		cout << "null\n";
		return;
	}
	else
		cout << "" << "Node {" << subroot->data.key << "|" << subroot->data.data <<
"}" << '\n';
	recursive_PrettyPrinted(subroot->left, indent + 1);
	recursive_PrettyPrinted(subroot->right, indent + 1);
}
```

```
ErrorCode Insert(NodeEntry<keyType, dataType> DataIn)
```

Chèn một NodeEntry vào cây nhị phân tìm kiếm

Nếu key của của NodeEntry này đã có sẵn trong cây nhị phân thì trả về: `ERRORCODE_DUPPLICATE`

Nếu không thì trả về kết quả: `ERRORCODE_SUCCESS`

```
ErrorCode Insert(NodeEntry<keyType, dataType> DataIn) {
	return recursive_Insert(_root, DataIn);
```

```
    }
    ErrorCode recursive_Insert(TreeNode<keyType, dataType> *&subroot,
NodeEntry<keyType, dataType> &DataIn) {
        //Your code here
        if (!subroot) {
            subroot = new TreeNode<keyType, dataType>(DataIn);
        }
        else if (DataIn.key < subroot->data.key)
        {
            recursive_Insert(subroot->left, DataIn);
        }
        else if (DataIn.key > subroot->data.key)
        {
            recursive_Insert(subroot->right, DataIn);
        }
        else return ERRORCODE_DUPPLICATE;
    }
```

`ErrorCode Search(NodeEntry<keyType, dataType> &DataOut, keyType target)`

Tìm kiếm một NodeEntry trong cây nhị phân tìm kiếm có key là target
Nếu tìm thấy trả về NodeEntry đó qua biến tham khảo DataOut và trả về kết quả: `ERRORCODE_SUCCESS`
Nếu không tìm thấy, trả về null cho biến tham khảo DataOut và trả về kết quả: `ERRORCODE_NOTFOUND`
Để thực hiện, SV dùng phương thức `recursive_Search` hoặc hàm `iterative_Search`

```
    ErrorCode Search(NodeEntry<keyType, dataType> &DataOut) {
        // Implemented by using recursive_Search OR iterative_Search
        // Your code here
        TreeNode<keyType, dataType> *pNode =  recursive_Search(_root,
DataOut.key);
        if (pNode == NULL)
            return ERRORCODE_NOTFOUND;
        DataOut.data = pNode->data;
        return ERRORCODE_SUCCESS;
    }
    TreeNode<keyType, dataType>* recursive_Search(const TreeNode<keyType,
dataType> *&subroot, keyType target) {
        // Your code here
        if (subroot == NULL || subroot->data.key = target)
            return subroot;
```

```
        else if (target < subroot->data.key)
            return recursive_Search(subroot->left, target);
        else if (target > subroot->data.key)
            return recursive_Search(subroot->right, target);
    }

    TreeNode<keyType, dataType>* iterative_Search(TreeNode<keyType, dataType>
*subroot, keyType target) {
        // Your code here
        while (subroot != NULL && subroot->data.key != target) {
            if (target < subroot->data.key)
                subroot = subroot->left;
            else
                subroot = subroot->right;
            return subroot;
        }
    }
```

`ErrorCode Delete(keyType target)`

Xoá một NodeEntry trong cây nhị phân tìm kiếm có key là target
Nếu tìm thấy xoá NodeEntry đó và trả về kết quả: `ERRORCODE_SUCCESS`
Nếu không tìm thấy, trả về kết quả: `ERRORCODE_NOTFOUND`
Để thực hiện, SV dùng phương thức `recursive_Delete` và `removeNode`

```
    ErrorCode Delete(keyType target) {
        // Implemented by using recursive_Delete AND iterative_Delete
        // Your code here
        return recursive_Delete(_root, target);
    }
    ErrorCode recursive_Delete(TreeNode<keyType, dataType> *&subroot, keyType
target) {
        // Your code here
        if (subroot == NULL)
            return ERRORCODE_NOTFOUND;
        else if (target < subroot->data.key)
            return recursive_Delete(subroot->left, target);
        else if (target > subroot->data.key)
            return recursive_Delete(subroot->right, target);
        else {
            removeNode(subroot, target);
```

```
            return ERRORCODE_SUCCESS;
        }
    }

    void removeNode(TreeNode<keyType, dataType> *&subroot, keyType target) {
        // Your code here
        TreeNode<keyType, dataType>* pDel = subroot;
        if (subroot->left == NULL)
            subroot = subroot->right;
        else if (subroot->right == NULL)
            subroot = subroot->left;
        else {
            TreeNode<keyType, dataType>* parent = subroot;
            pDel = parent->left;
            while (pDel->right != NULL) {
                parent = pDel;
                pDel = pDel->right;
            }
            subroot->data = pDel->data;
            if (parent == subroot)
                parent->left = pDel->left;
            else
                parent->right = pDel->left;
        }
        delete pDel;
    }
```


Phần 3: Yêu cầu

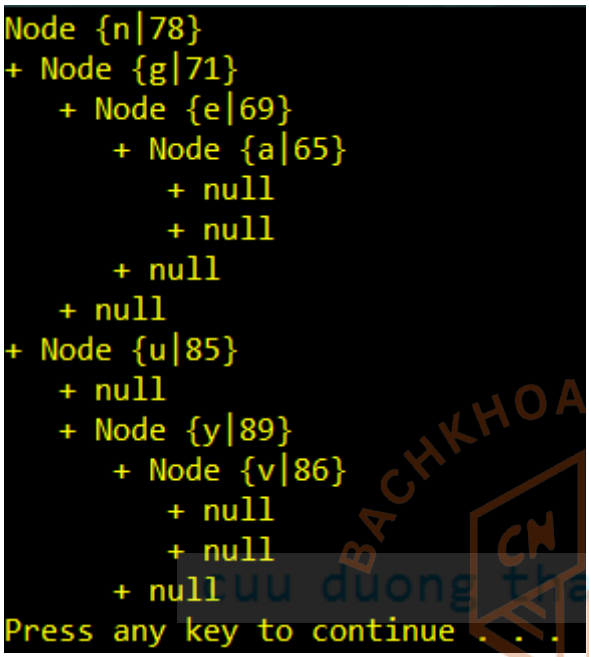
1. SV thực hiện (implement) tất cả các hàm được chú thích "// Your code here" trong file **BinarySearchTree.h**
2. Ứng dụng:

   SV tự tạo một cây nhị phân với các NodeEntry có
   - key: là từng kí tự (char) viết thường trong họ tên của SV

- data: là số nguyên (int), mang giá trị là mã ASCII từng kí tự viết hoa tương ứng với kí tự ở key

VD: SV tên Nguyễn Văn A, ta có “nguyenvana”

Tạo được cây như sau:



HẾT